# Blocked Band Reduction for Symmetric and Unsymmetric Matrices

SIVASANKARAN RAJAMANICKAM and TIMOTHY A. DAVIS
University of Florida

We introduce blocked algorithms for bidiagonal and tridiagonal reduction of banded matrices. The algorithms reduce a block of entries via pipelined plane rotations with efficient cache reuse that result in much less fill-in than Householder transformations. The blocked plane rotations can be simultaneously applied to an identity matrix as part of the singular value decomposition, with a method that exploits the non-zero pattern of the matrix as the rotations are applied. Our methods are several times faster than existing methods, while using much less memory.

Categories and Subject Descriptors: G.4 [**Mathematics of Computing**]: Mathematical Software—*algorithm analysis, efficiency*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Band Reduction, Band Matrices, Blocked Givens Rotations, Bidiagonal reduction, Tridiagonal Reduction

## 1. OVERVIEW

Eigenvalue computations of symmetric band matrices depend on a reduction to tridiagonal form [Golub and VanLoan 1996]. Given a symmetric band matrix $A$ of size $n$-by-$n$ with $l$ subdiagonals and super diagonals, the tridiagonal reduction can be written as

$$U^T A U = T$$

where $T$ is a tridiagonal matrix and $U$ is orthogonal. Singular value decomposition of an $m$-by-$n$ unsymmetric band matrix $A$ with $l$ subdiagonals and $u$ super diagonals depends on a reduction to bidiagonal form. We can write the bidiagonal reduction of $A$ as

$$U^T A V = B$$

where $B$ is a bidiagonal matrix and $U$ and $V$ are orthogonal.

Rutishauser [Rutishauser 1963] gave the first algorithm for band reduction using a pentadiagonal matrix. This algorithm removes the fill caused by a plane rotation

Dept. of Computer and Information Science and Engineering and Univ. of Florida, Gainesville, FL, USA. email: davis@cise.ufl.edu. http://www.cise.ufl.edu/∼davis. Dept. of Computer and Information Science and Engineering and Univ. of Florida, Gainesville, FL, USA. email: srajaman@cise.ufl.edu. http://www.cise.ufl.edu/∼srajaman. This work was supported by the National Science Foundation, under grants 0620286, 0619080, and 0203270.

[Givens 1958] immediately with another rotation. Schwarz generalized this method for tridiagonalizing a symmetric band matrix [Schwarz 1968; 1963]. Both algorithms use plane rotations (or Jacobi rotations) to zero one non-zero entry of the band resulting in a scalar fill. The algorithms then use new plane rotations to reduce the fill, creating new fill further down the matrix, until there is no fill. This process of moving the fill down the matrix is called chasing the fill.

[Kaufman 1984] suggested an approach to reduce and chase more than one entry at a time using vector operations. The algorithm uses plane rotations and reduces one diagonal at a time. This is the method in the LAPACK [Anderson et al. 1999] library for bidiagonal and tridiagonal reduction (xGBBRD and xSBTRD routines). The latest symmetric reduction routines in LAPACK use a revised algorithm by Kaufman [Kaufman 2000]. Though Kaufman's algorithms reduce more than one entry at a time it is not a blocked algorithm as each of the entries has to be separated by a distance. [Rutishauser 1963] also suggested using Householder tranformations for band reduction and chasing the triangular fill. [Murata and Horikoshi 1975] also used Householder transformations to introduce the zeroes in the band but chose to chase only part of the fill. This idea reduced the floating point operations (flops) required but increased the workspace. Recently, [Lang 1993] used this idea for parallel band reduction. The idea is also used in the SBR tool box [Bischof et al. 2000b]. However, they use QR factorization of the subdiagonals and store it as $WY$ transformations [Van Loan and Bischof 1987] so that they can take advantage of level 3 style BLAS operations. The SBR framework can choose to optimize for floating point operations, available work space and using the BLAS.

Some of the limitations of existing methods can be summarized as follows:

—they lead to poor cache access and they use expensive non-stride-one access row operations during the chase (or)

—they use more memory and/or more work to take advantage of cache friendly BLAS3 style algorithms.

We propose an algorithm that uses plane rotations, blocks the rotations for efficient use of cache, does not generate more fill than non blocking algorithms, performs only slightly more work than the scalar versions, and avoids non-stride-one access as much as possible when applying the blocked rotations. [Rutishauser 1963] discussed blocking plane rotations in the context of Givens method [Givens 1958] for reducing a full symmetric matrix to tridiagonal form. We discuss the similarity between our method and this idea in section 2.

When we want to compute $U$ or $V$ in the above equations we can start with an identity matrix and apply the plane rotations to it. There are two types of methods to do this: we apply the plane rotations when they are applied to the original band matrix (forward accumulation) or we save all the rotations and apply it to the identity matrices later (backward accumulation). Even though simple forward accumulation requires more floating point operations than backward accumulation, [Kaufman 1984] showed that we can do forward accumulation efficiently by exploiting the non zero pattern when applying the rotations to the identity. Kaufman's accumulation algorithm will do the same number of floating point operations as backward accumulation. The non zero pattern of $U$ or $V$ will be dependent upon the order in which we reduce the entries in the original band matrix. Our accumu-

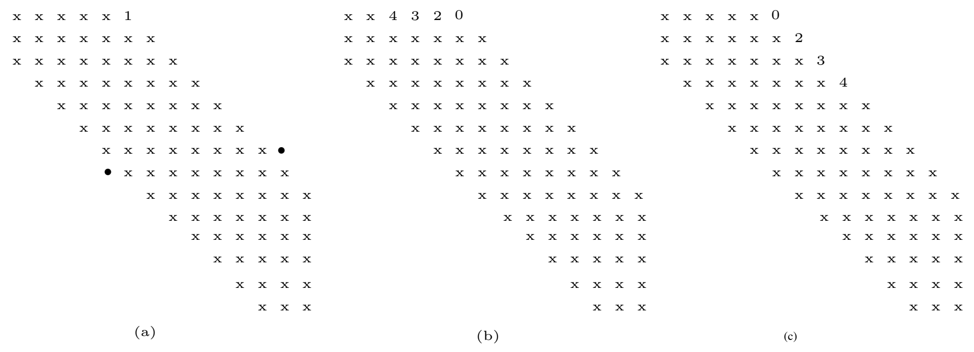

Fig. 1. (a) First iteration in Schwarz's band reduction. (b) Schwarz's row reduction method (c) Schwarz's diagonal reduction method.

lation algorithm exploits the non zero pattern of $U$ or $V$ while using our blocked reduction.

Section 2 introduces the blocking idea. Blocking combined with pipelining the rotations helps us achieve minimal fill, as discussed in in section 3. We introduce the method to accumulate the plane rotations efficiently in section 4. The performance results for the new algorithms are in section 5.

We refer to a plane rotation between two columns (rows) as column (row) rotations. We use an unsymmetric band matrix as an example even though the original algorithms are for the symmetric band case as our algorithm applies to both symmetric and unsymmetric matrices. We refer to a band matrix with $l$ subdiagonals and $u$ super diagonals as an $(l, u)$*-band matrix.* The term $k$*-lower Hessenberg matrix* refers to a matrix with $k$ super diagonals and the lower triangular part.

## 2. BLOCKING FOR BAND REDUCTION

We give a brief idea of Schwarz's generalizations [Schwarz 1968][Schwarz 1963] of Rutishauser's method [Rutishauser 1963] as our blocked method for band reduction uses a mixed approach of these two methods.

A step in Schwarz's algorithms is reducing an entry in the band and chasing the resultant fill down the matrix. Figure 1(a) shows the first step in the reduction of a 14-by-14 $(2, 5)$-band matrix when using Schwarz's methods. The first plane rotation tries to reduce the entry marked with 1. This causes a fill in the third subdiagonal (marked as $\bullet$). The methods rotate rows 7 and 8 and columns 12 and 13 to complete the chase. However, Schwarz's methods differ in whether it reduces an entire row to bidiagonal first ([Schwarz 1968]) or it reduces one diagonal at a time ([Schwarz 1963]) in the subsequent steps. We will call the former the row reduction method and the later the diagonal reduction method. Figure 1(b)-(c) shows the difference between the two methods. In steps $2 \dots 4$, the row reduction method will reduce the entries marked $2 \dots 4$ in figure 1(b). In steps $2 \dots 4$, the diagonal reduction method will reduce the entries marked $2 \dots 4$ in figure 1(c).

Given a symmetric $(b, b)$-band matrix $A$ of the order $n$, we can write the two algorithms of Schwarz as:

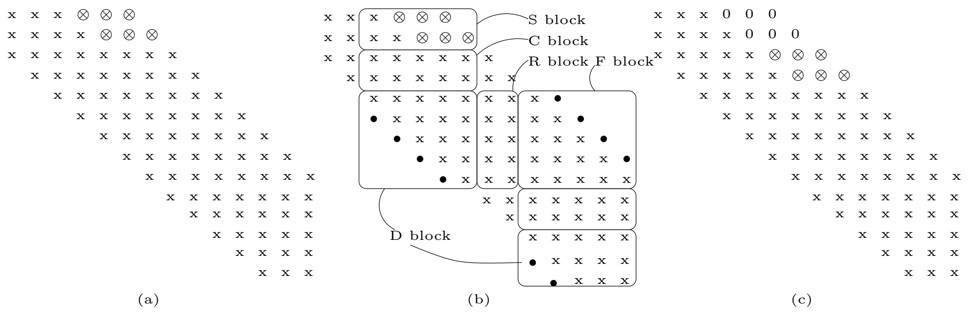

Fig. 2. Blocked band reduction. (a) First iteration of the reduction. (b) Blocks for the first iteration of the reduction. (c) Second iteration of the reduction.

**Schwarz's row reduction algorithm for symmetric band reduction**

```
for i = 1 : n-2
    for j = MIN(i+b, n)-1 : -1 : i+1
        Rotate columns j and j+1 to reduce A[i, j+1].
        Chase the fill.
```

**Schwarz's diagonal reduction algorithm for symmetric band reduction**

```
for j = b-1 : -1 : 1
    for i = 1 : n-(j+1)
        Rotate columns i+j and i+j+1 to reduce A[i, i+j+1].
        Chase the fill.
```

Schwarz's band reduction algorithms are not blocked algorithms. Rutishauser [Rutishauser 1963] modified the Givens method [Givens 1958] for reducing a full symmetric matrix to tridiagonal form to use blocking. The structure of the block, a parallelogram, will be the same for the modified Givens method and our algorithm, but we will allow the number of columns and number of rows in the block to be different. Furthermore, [Rutishauser 1963] led to a bulge-like fill which is expensive to chase, whereas our method results in less intermediate fill which takes much less work to chase.

Our blocking scheme combines Schwarz's row reduction method [Schwarz 1968] and Rutishauser's modified Givens method [Rutishauser 1963]. A step in our blocked reduction algorithm reduces an $r$-by-$c$ block of entries and chases the resultant fill. At each step, we choose an $r$-by-$c$ parallelogram as our block such that its reduction and the chase avoids the bulge-like fill when combined with our pipelining scheme (given in section 3). The first $c$ entries reduced by the row reduction method and the first $r$ entries reduced by the diagonal reduction method are the outermost entries of the parallelogram for the first step in the blocked reduction. Figure 2(a) shows the selected block for the first step when the block is of size 2-by-3.

At each step, once we select the block, we partition the entries in the band that change in the current step into blocks. The blocks can be one of five types:

*S-block.* The S (*seed*) block consists of the $r$-by-$c$ parallelogram to be reduced and the entries in the same $r$ rows as the parallelogram to which the rotations are applied. The S-block in the upper(lower) triangular part is reduced with column(row) rotations.

*C-block.* The C (*column rotation*) block has the entries that are modified only by column rotations in the current step.

*D-block.* The D (*diagonal*) block consists of the entries that are modified by both column and row rotations (in that order) in the current step. Applying column rotations in this block causes fill which are chased by new row rotations. This is at most a $(c + r)$-by-$(c + r)$ block.

*R-block.* The R (*row rotation*) block has the entries that are modified only by row rotations in the current step.

*F-block.* The F (*fill*) block consists of the entries that are modified by both row and column rotations (in that order) in the current step. Applying row rotations in this block causes fill which is chased by new column rotations. This is at most a $(c + r)$-by-$(c + r)$ block.

Generally the partitioning results in one seed block and potentially more than one of the other blocks. The blocks other than the seed block repeat in the same order. The order of the blocks is C, D, R and F-block when the seed block is in the upper triangular part. Figure 2(b) shows all the different blocks for the first step. The figure shows more than one fill. But no more than one fill is present at the same time, so only a single scalar is required to hold all the fill. The pipelining (explained below) helps us restrict the fill. As a result, our method uses very little workspace other than the matrix being reduced and an $r$-by-$c$ data structure to hold the coefficients for the pending rotations being applied in a single sweep.

Once we complete the chase in all the blocks, subsequent steps reduce the next $r$-by-$c$ entries in the same set of $r$ rows as long as there are more than $r$ diagonals left in them. This results in the following two pass algorithm where we reduce the bandwidth to $r$ first and then reduce the thin band to bidiagonal matrix in the next phase. Figure 2(c) highlights the entries for the second step with $\otimes$. The two pass algorithm is listed below.

**Algorithm for band reduction**

```
while $lowerbandwidth > 0$ or $upperbandwidth > 1$
    for each set of r rows/columns
        reduce the band in lower triangular part
        reduce the band in upper triangular part
    set $c$ to $r - 1$
    set $r$ to 1.
```

There are two exceptions with respect to the the number of diagonals left. The number of diagonals left in the lower triangular part is $r - 1$ instead of $r$ when $r$ is one (to avoid the second iteration of the while loop in the above algorithm) and when the input matrix is an unsymmetric $(l, 1)$-band matrix for more predictable accumulation of the plane rotations. See section 4 for more details on the latter.

The algorithm so far assumes the number of rows in the block for upper triangular part and the number of columns in the block for the lower triangular part is the same. We reduce the upper and lower triangular part separately if that is not the case. The algorithm for the band reduction in the upper triangular part is given below.

**Algorithm for band reduction in upper triangular part**

```
for each set of $c$ columns in current set of rows
    Find the trapezoidal set of entries to zero in this iteration
    Divide the matrix into blocks
    Find column rotations to zero $r$-by-$c$ entries in the S-block and
    apply them in the S-block
    While there are column rotations
        Apply column rotations to the C-block
        Apply column rotations to the D-block, Find row rotations to chase fill
        Apply row rotations to the R-block
        Apply row rotations to the F-block, Find column rotations to chase fill
        Readjust the four blocks to continue the chase.
```

The algorithm for reducing the lower triangular part is similar to the one for the upper triangular part except that the order of the blocks in the chasing is different. In the inner while loop the rotations are applied to the R, F, C and D-blocks, in that order.

## 3. PIPELINING THE GIVENS ROTATIONS

The algorithm for the reduction of the upper triangular part in section 2 relies on finding and applying the plane rotations on the blocks. Our pipelining scheme does not cause triangular fill-in and avoids non-stride-one access of the matrix. The general idea is to find and store $r$-by-$c$ rotations in the workspace and apply them as efficiently as possible in each of the blocks to extract maximum performance. Figures 3 - 7 in this section show the first five blocks from Figure 2(b) and the operations in all of them are sequentially numbered from 1..70. In the figures, a solid arc between two entries represents the generation of a row (or column) rotation between adjacent rows (or columns) of the matrix to annihilate one of the two entries. A dotted arc represents the application of these rotations to other pairs of entries in the same pair of rows (or columns) of the matrix.

### 3.1 Seed Block

The $r$-by-$c$ entries to be reduced in the current step are in the S-block. The number of rows in this block is $r$ and the number of columns is at most $c+r$. We find new column rotations to reduce the entries one row at a time. The rotations generated from the entries in the same row (column) in the upper (lower) triangular part are called a wave. At any row in the seed block, we apply rotations from previous rows before finding the new rotations to reduce the entries in the current row. This is illustrated in the figure 3. We reduce the three entries (marked as $\otimes$) in the first row with column rotations $G_1 \ldots G_3$ and save the rotations in work space. This

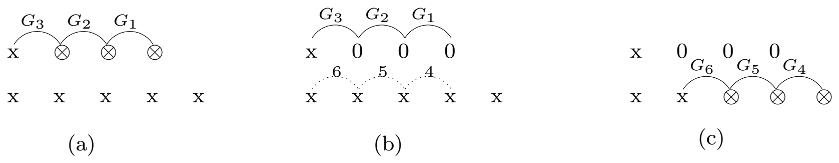

Fig. 3. Column rotations in S-block in the upper band: (a) Generate column rotations in the first row (b) Apply column rotations to the second row (c) Generate column rotations in the second row

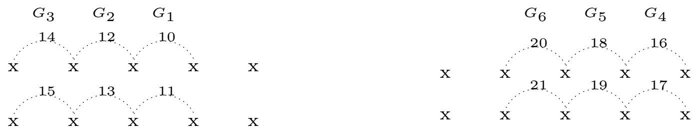

Fig. 4. Applying column rotations to C-block.

wave of rotations is also applied to the second row as operations $4 \dots 6$. We then reduce the three entries in the second row with new set of rotations $G_4 \dots G_6$ and save the rotations.

The algorithm to generate and apply the rotations in the S-block is listed below.

**Generating and applying rotations in the S-block**

```
For all rows in the S-block
    If not the first row apply column rotations from the previous rows.
    Find the column rotations to zero entries in current row (if there are any)
```

Finding and applying rotations in the S-block is the only place in the algorithm where we use explicit non-stride-one access to access the entries across the rows.

## 3.2 Column Block

We apply $r$ waves of rotations to the C-block. Each wave consists of $c$ column rotations. We apply each column rotation to the entire C-block one at a time (and one wave at a time), as this would lead to efficient stride-one access of memory, assuming the matrix is stored in column-major order.

A column rotation operates on pairs of entries in adjacent columns (columns $i-1$ and $i$, say). The next column rotation in this wave operates on columns $i-2$ and $i-1$, reusing the $i-1$th column which presumably would remain in cache. Finally, when this first wave is done we repeat and apply all the remaining $r-1$ waves to the corresponding columns of the C-block. The column rotations in any wave $j+1$ will use all but two columns used by the wave $j$ leading to possible cache reuse. Figure 4(a)-(b) illustrates the order in which we apply the two waves of rotations ($G_1 \dots G_3$ and $G_4 \dots G_6$) to the 2 rows in the C-block. The algorithm for applying the rotations in the C-block is given below.

**Applying rotations in the C-block**

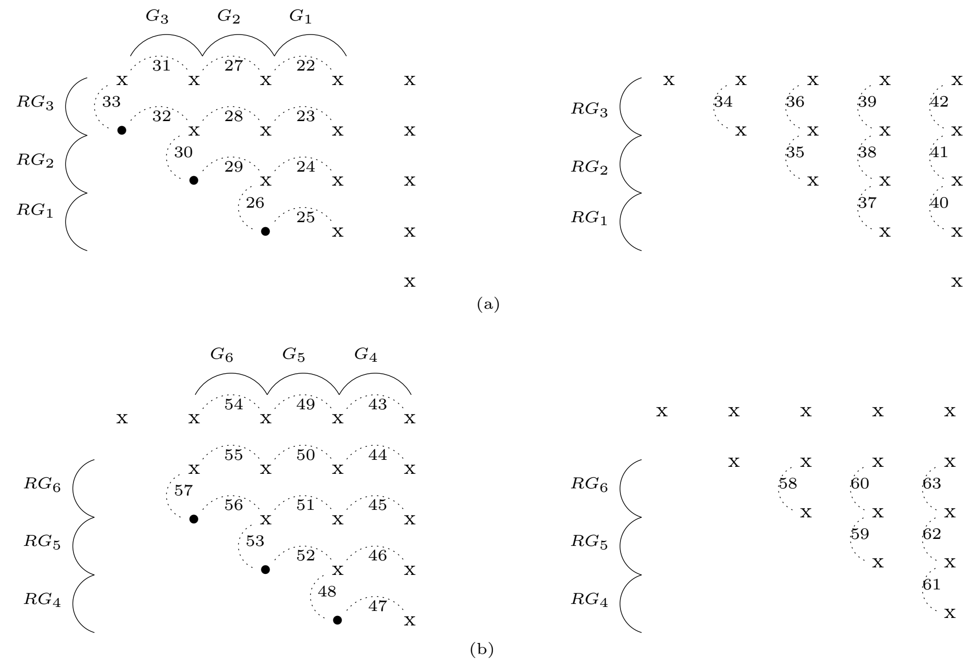

Fig. 5. Applying column and row rotations to D-block.(a) Two phases for the first wave of rotations. (b) Two phases for the second wave of rotations.

```
for each wave of column rotations
    for each column in this wave of rotations
        Apply the rotation for the current column from current wave to all the
        rows in the C-block.
    Shift column indices for next wave.
```

## 3.3 Diagonal Block

We apply $r$ waves each with $c$ column rotations to the D-block. Each column rotation $G_i$ (say between columns $k$ and $k-1$) can generate a fill in this block. To restrict the fill to a scalar, we reduce the fill immediately with a row rotation ($RG_i$) before we apply the next column rotation $G_{i+1}$ (to columns $k-1$ and $k-2$). We save the row rotation after reducing the fill and continue with the next column rotation $G_{i+1}$, instead of applying the row rotation to the rest of the row as that would lead to non-stride-one access. When all the column row rotations in the current wave are done, we apply the new wave of row rotations to each column in the D-block.

This algorithm leads to two sweeps on the entries of the D-block for each wave of column rotations: one from right to left when we apply column rotations that generate fill, find row rotations, reduce the fill and save the row rotations, and the next from left to right when we apply the pipelined row rotations to all the columns in the D-block. We repeat the two sweeps for each of the subsequent $r-1$ waves

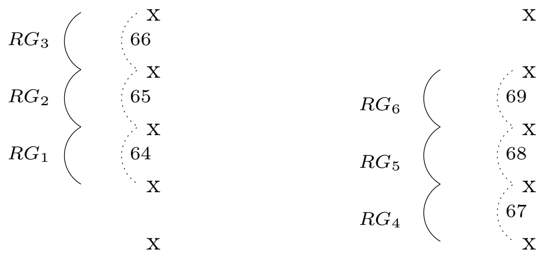

Fig. 6. Applying row rotations to one column in R-block.

of column rotations. Figure 5(a) shows the order in which we apply the column rotations $G_1 \dots G_3$, find the corresponding row rotations and apply them to our example. The operations $22 \dots 42$ show these steps. Operations $22 \dots 33$ show the right to left sweep and $34 \dots 42$ show the left to right sweep.

Figure 5(b) shows the order in which we apply a second wave of rotations $G4 \dots G6$ to our example and handle the fill generated by them (numbered $43 \dots 63$). The algorithm for applying and finding rotations in the D-block is listed below.

**Applying and finding rotations in the D-block**

```
for each wave of column rotations
    for each column in this wave of rotations (right-left)
        Apply the rotation for the current column from current wave to all the
        rows in the D-block, generate fill.
        Find row rotation to remove fill.
        Apply row rotation to remove fill. (not to entire row)
    for each column in this wave of rotations (left-right)
    and additional columns in the D-block to the right of these columns
        Apply all pending row rotations to current column.
    Shift column indices for next wave.
```

## 3.4 Row Block

We apply $r$ waves of rotations to the R-block. Each wave consists of $c$ row rotations. We could apply each row rotation to the entire R-block one at a time (and one wave at a time), but this would require an inefficient non-stride-one access of memory, assuming the matrix is stored in column-major order.

Instead, we apply all the $c$ row rotations to the just the first column of the R-block, for the first wave. A row rotation operates on adjacent entries in the column (rows $i-1$ and $i$, say). The next row rotation in this wave operates on rows $i-2$ and $i-1$, reusing the cache for entry from row $i-1$. Finally, when this first wave is done we repeat and apply all the remaining $r-1$ waves to the first column of the R-block, which presumably would remain in cache for all the $r$ waves. This entire process is then repeated for the second and subsequent columns of the C-block. This order of operations leads to efficient cache reuse and purely stride-one access of the memory in the R-block.

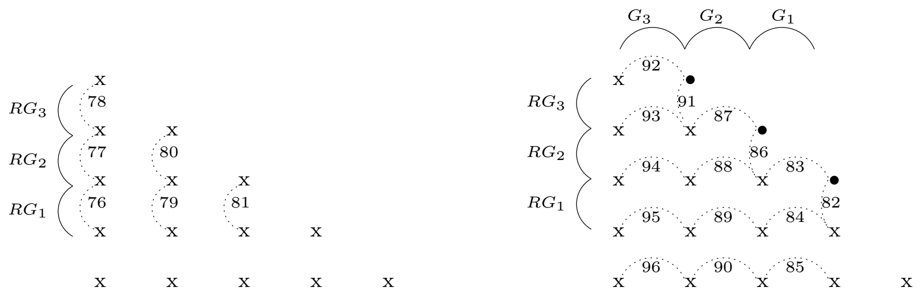

Fig. 7. Applying row and column rotations to F-block.

Figure 6 shows two waves of row rotations $RG_1 \dots RG_3$ and $RG_4 \dots RG_6$ applied to one column in the R-block. The order in which we apply the rotations to the column is specified by numbers $64 \dots 69$. The algorithm for applying the rotations in the R-block is given below.

**Applying rotations in the R-block**

```
for each column in the R-block
    for each wave of row rotations
        Apply all the rotations in current wave to appropriate rows
        in current column.
        Shift row indices for next wave.
```

There is no fill in the R-block. We do not show the operations $70 \dots 75$ in the next column of the R-block (shown in figure 2(b)) in the figure 6. But it will have the same pattern as the column shown.

## 3.5 Fill Block

We apply $c$ row rotations from each wave to the F-block and generate one wave of column rotations after reducing the fill. This set of operations is like a mirror image of the column operations in the D-block. We follow the same two pass approach for each wave of rotations, but use a different order for applying the rotations to take advantage of the cache. In the first pass, for each column in the F-block we will apply all the row rotations in the current wave but stop short of creating any fill. The second pass is a right to left pass of all the columns in the F-block where we will generate the fill in that column by applying the row rotation, find a column rotation to remove the fill, and apply the column rotation to the rest of the column. Figure 7 shows the order in which we apply the row rotations (operations $76 \dots 81$),generate fill (operations $82, 86, 91$), and apply the column rotations. Operations $76 \dots 81$ constitute the first pass. Operations $82 \dots 96$ constitute the second pass. We can handle multiple waves of rotations just by shifting the indices and following the two pass algorithm for each of the waves of the rotations. The algorithm for applying the rotations in the F-block is given below.

**Applying and finding rotations in the F-block**

```
for each wave of row rotations
    for each column in the F-block (left-right)
        Apply all rotations from current wave, except those that generate
        fill-in in the current column, to current column.
    for each column in the F-block (right-left)
        Apply the pending rotation that will create fill.
        Find column rotation to remove fill.
        Apply column rotation to entire column.
    Shift row indices for next wave.
```

To summarize, the only non-stride one access we do in the entire algorithm is in the seed block when we try to find the rotations. We generate no more fill than the scalar algorithms of Schwarz. We do slightly more work than row reduction method but less than diagonal reduction method for a typical block size. This is due to the two passes of our algorithm at the top level.

## 3.6 Flops

For the purposes of computing the exact floating point operations, we assume that finding the plane rotations takes six operations, as in the real case with no scaling of the input parameters (two multiplies, two divides, one addition and a square root). We also assume that applying the plane rotations to two entries requires six floating point operations as in the real case (four multiplies and two additions).

Let us consider the case of reducing $(u, u)$-symmetric band matrix of order $n$ to tridiagonal form by operating only on the upper triangular part of the symmetric matrix.

The number of floating point operations required to reduce the symmetric matrix using Schwarz's diagonal reduction method is

$$f_d = 6 \sum_{d=3}^{b} \sum_{i=d}^{n} d + 2 \left\lfloor \frac{\max(i-d, 0)}{d-1} \right\rfloor (d+1) + \text{rem}\left( \frac{\max(i-d, 0)}{d-1} \right) + 2$$

where rem is the remainder function and $b = u + 1$. The number of floating point operations required to reduce the symmetric matrix using Schwarz's row reduction method is

$$f_r = 6 \sum_{i=1}^{n-2} \sum_{d=3}^{\min(b, n-i+1)} d + 2 \left\lfloor \frac{\max(i-d, 0)}{b-1} \right\rfloor (b+1) + \text{rem}\left( \frac{\max(i-d, 0)}{b-1} \right) + 2$$

For finding the number of of floating point operations for our blocked reduction, when block size is $r$-by-$c$ we note that we perform the same number of operations as using a two step Schwarz's row reduction method where we first reduce the matrix to $r$ diagonals using Schwarz's row reduction method and then reduce the $r$ diagonals using Schwarz's row reduction again. Then we can obtain the floating point operations required by the blocked reduction to reduce the matrix by using

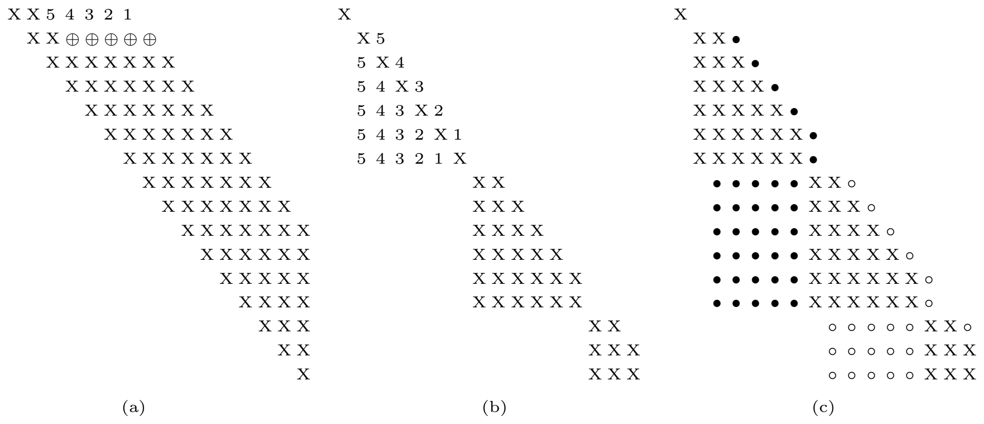

Fig. 8. Update of $U$.(a) Upper triangular part of symmetric matrix $A$ (b) $U$ after reducing entries $1 \ldots 5$ (c) $U$ after reducing $\oplus$ entries.

the equation for $f_r$ from above.

$$f_{b1} = \sum_{i=1}^{n-r-1} \sum_{d=r+2}^{\min(b,n-i+1)} d + 2 \left\lfloor \frac{\max(i-d,0)}{b-1} \right\rfloor (b+1) + \operatorname{rem}\left( \frac{\max(i-d,0)}{b-1} \right) + 2$$

$$f_{b2} = \sum_{i=1}^{n-2} \sum_{d=3}^{\min(r+1,n-i+1)} d + 2 \left\lfloor \frac{\max(i-d,0)}{r} \right\rfloor (r+2) + \operatorname{rem}\left( \frac{\max(i-d,0)}{r} \right) + 2$$

$$f_b = 6\,(f_{b1} + f_{b2})$$

Asymptotially, all three methods result in the same amount of work. But the work differs by a constant factor. All three methods require the same amount of work when $l < 3$. Given a 1000-by-1000 matrix, and $l = 3$, Schwarz's row reduction method requires only 90% floating point operations as of diagonal reduction method. Depending on the block size, the flops required by the blocked method will be equal to one of the other two methods. When we increase the bandwidth to $l = 150$ for the same matrix, the row reduction method requires 10% less flops than the diagonal reduction method and 3% less flops than the blocked reduction method with default block size. When the input matrix is dense, $l = 999$, the row reduction method require 25% less flops than the diagonal reduction method and 5% less flops than then blocked method with default block size. To summarize, the row reduction method in this example leads to the least flops, but the blocked reduction is only 3-5% worse in terms of the flops. We will show that this can be compensated by the performance advantage of blocking.

## 4. ACCUMULATING THE PLANE ROTATIONS

Our blocked approach to the band reduction leads to a different non zero pattern in $U$ and $V$ than the one described by [Kaufman 2000]. In this section, we will describe the fill pattern in $U$ for different block sizes for our blocked reduction

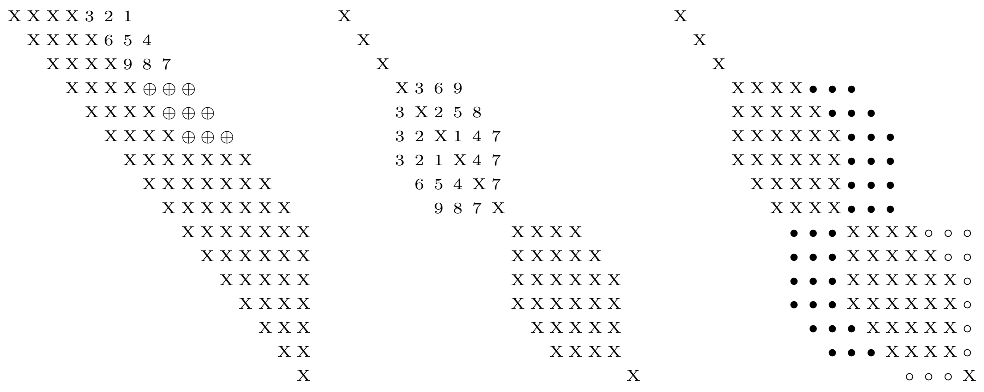

Fig. 9. Update of $U$. (a) Upper triangular part of symmetric matrix $A$ with $r = 3$ (b) $U$ after reducing entries $1 \ldots 9$ (c) $U$ after reducing $\oplus$ entries.

method. We will omit $V$ from the discussion hereafter though the entire disussion applies to $V$ too.

Given a band matrix $A$ and a $r$-by-$c$ block, the algorithm to find the non zero pattern in $U$ differs based on four possible cases

—$A$ is a symmetric or $A$ is an unsymmetric (0, u)-band matrix.

—$A$ is an unsymmetric (l, 1)-band matrix.

—$A$ is an unsymmetric $(l, u)$-band matrix where $l > 0$, $u > 1$ and $r = 1$.

—$A$ is an unsymmetric $(l, u)$-band matrix where $l > 0$, $u > 1$ and $r > 1$.

The structure of the algorithm is the same in all four cases. The reduction of first $r$ columns and/or rows, will lead to a specific non zero pattern in the upper and lower triangular part of $U$. The nonzero pattern caused by rest of the plane rotations will depend upon the nonzero pattern from the reduction of first $r$ columns and/or rows.

The accumulation algorithm uses the fact that applying plane rotations between any two columns in $U$ will result in the non zero pattern of both the columns to be equal to the the union of the original non zero pattern of each of the columns. Given an identity matrix, when we apply the plane rotations in a particular order, as dictated by the blocked reduction, we can track the first and last non zero entries of any column in $U$ for applying the rotations only to the nonzeros. It will be easy to do this for $U$ and $V$ if we have two arrays of size $n$. But the idea below uses constant space to do the same. We describe the non zero pattern for all the four cases below.

**Symmetric** $A$: We consider both a symmetric $(u, u)$-band matrix and an unsymmetric $(0, u)$-band matrix as equivalent in this section. The reduction of both these matrices as symbolically equivalent, as we operate only on the upper triangular part of the symmetric matrix. When $A$ is a symmetric $(u, u)$-band matrix and $r = 1$, our reduction algorithm in section 2 will reduce each row to tridiagonal form. The blocked reduction then results in $u - 1$ plane rotations applied to $u$ consecutive columns of $U$, say columns $2 \ldots u + 1$ when reducing the the first row to tridiagonal. The block fill in $U$ for this set of columns will be a lower Hessenberg

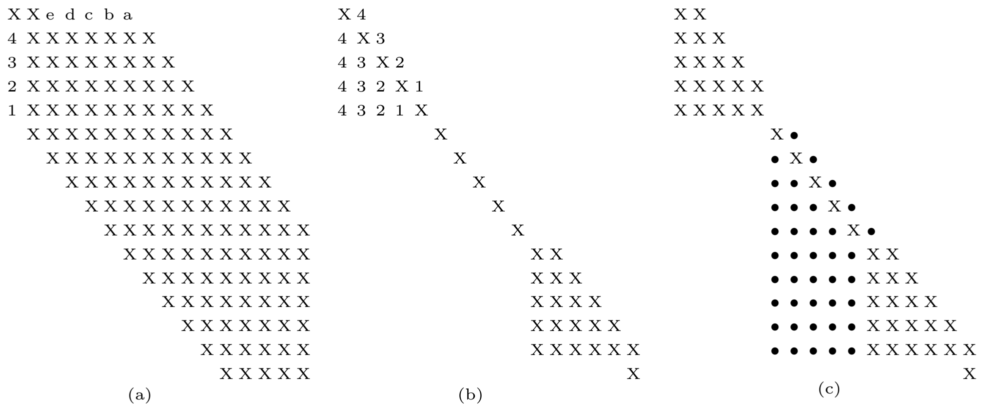
Fig. 10. Update of $U$. (a) Unsymmetric $A$ with $r = 1$ (b) $U$ after reducing the entries $1 \dots 4$. (c) $U$ after reducing the entries a. . .e.

matrix of the the order of $u$. The fill in $U$, when chasing the fill further down the matrix, will be in the subsequent $u$ columns of U and will have the same lower Hessenberg structure. At the end of the chase, $U$ will be block diagonal, where each block is lower Hessenberg of order $u$. For example, consider reducing the first row of entries marked $1 \dots 5$ in the figure 8(a). Figure 8(b) shows the fill pattern after applying to $U$ the plane rotations that were generated to reduce these entries and chase the resultant fill. The first block of fill in $U$, generated by chasing the fill when we reduce entry marked 1 in figure 8(a) is marked 1 too. The fill generated by the plane rotations during further chase are just marked $x$.

Once the chase for the reducing the first iteration is complete every other row will result in a u-by-(u-1) block fill in the lower traingular part and an increase in one super diagonal. Figure 8(c) shows the fill in $U$ caused by reducing the second row of $A$ and by chasing the resultant fill. The first block of the fill is marked with $\bullet$ and the fill in other blocks are marked with $\circ$. Note that keeping track of the fill in the upper triangular is relatively easy as all plane rotations that reduce entries from row $k$ and chase the resultant fill will generate fill in the k-th super diagonal for any column in $U$. This fill pattern in $U$ is exactly same as the fill pattern if we reduced $A$ with Schwarz's row reduction method, as blocked reduction will follow the same order of rotations when $r$ is one.

When $r > 1$ the each block in the block diagonal $U$, after reducing the first $r$ rows, will be a $(u - r, r)$-band matrix of the order $u$. Each subsequent reduction of $r$ rows and their chase will generate a block fill in $U$ that consists of a (u-r+1)-by-(u-r) block and a below that a block of (r-1)-by-(u-1) $(0, u - r)$-band matrix with a zero diagonal in the lower triangular part. As discussed in the $r = 1$ case, there will also be an addition of $r$ super diagonals in $U$, for every $r$ rows that are reduced in $A$. Figure 9 shows the fill in $U$ for the first two iterations when $r = 3$. The conventions are same as before. For the first iteration the fill is numbered for the entries in $A$ that caused the fill, and further fill from the chase are shown as $x$. For the second iteration the two different blocks of fill are differentiated with $\bullet$ and $\circ$.

$(l, 1)$**-band** $A$: Given a $(l, 1)$-band matrix the band reduction algorithm will

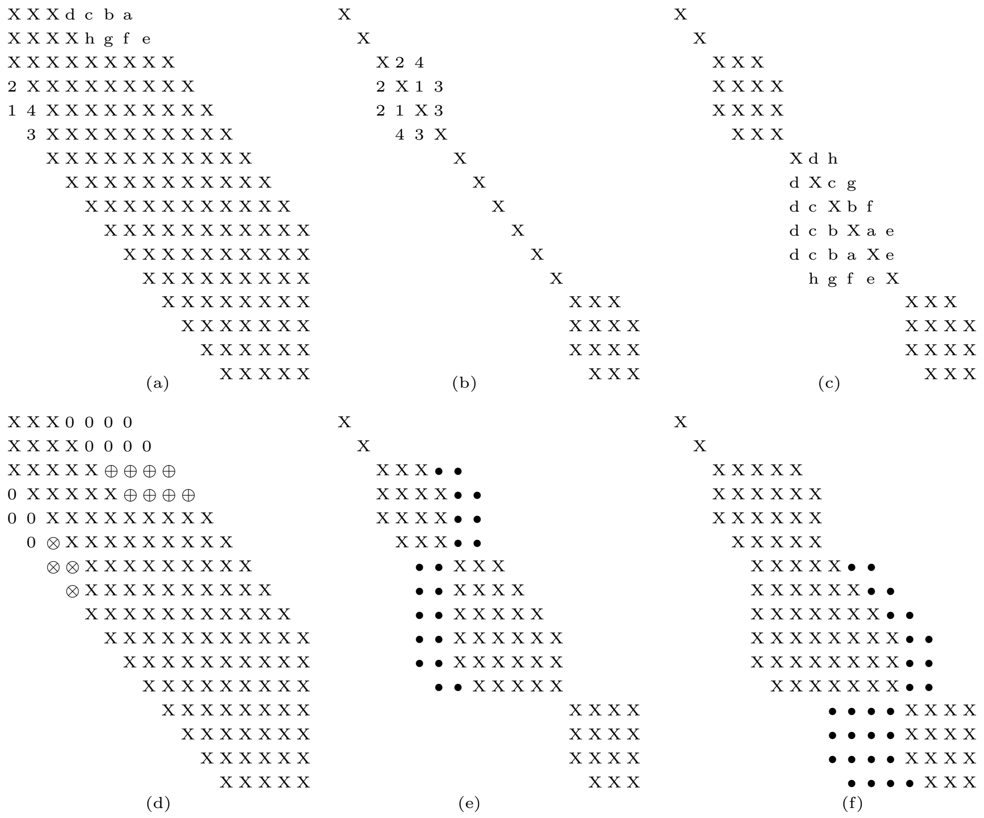

Fig. 11. Update of U (a) Unsymmetric $A$ with $r = 2$ (b) $U$ after reducing entries $1 \ldots 4$ (c) $U$ after reducing entries a…h (d) $A$ after first two iterations (e) $U$ after reducing $\otimes$ entries. (f) $U$ after reducing $\oplus$ entries.

reduce the lower triangular part to $r-1$ diagonals to keep the pattern of the fill in $U$ simple. When $r = 1$, reducing the first column of $A$ and chasing the fill will lead to a block diagonal $U$, each block lower Hessenberg of the order $l+1$. Each subsequent column in $A$, when reduced, will result in rectangular blocks of fill of the order of (l+1)-by-l in the lower triangular part of $U$, and one additional super diagonal in the upper triagular part of $U$. The basic structure of the fill will be similar to the one shown in figure 8(b)-(c) except the size of the blocks will be different.

When $r > 1$, reducing the first $r$ columns of $A$ will result in block diagonal $U$, where each block will be $(l-r+2, r)$-band of the order $(l+1)$. Each subsequent $r$ set of columns of $A$ will result in a block of fill that consists of a (l-r+2)-by-(l-r+1) block and below that a block in the shape of $(0, l-r+1)$-band of the order (r-1)-by-l with a zero diagonal in the lower triangular part of $U$, and $r$ additional super diagonals in the upper triangular part of $U$. The basic structure of the fill will be similar to the one shown in figure 9(b)-(c) except the size of the blocks will be different.

**Unsymmetric $A$, $r = 1$:** When we consider the case of the unsymmetric $(l, u)$-band matrix $A$, to find the non zero pattern of $U$, we need to only look at the interaction between the plane rotations to reduce the entries in lower triangular part and the plane rotations to chase the fill from lower triangular part (even though the original entries were from the upper triangular part), as these will be the only rotations applied to $U$. We can simply say that we need to look at all the row rotations to matrix $A$. Let us consider the case when the the number of rows in the block $r$ is one first. The reduction algorithm from section 2 will reduce the first column and row pair to the bidiagonal form before reducing subsequent column and rows.

The plane rotations to reduce the entries in the first column will be between rows $l+1 \ldots 1$ resulting in a lower Hessenberg block of the size (l+1)-by-(l+1) in $U$ as we saw in the symmetric case. The plane rotations from the resultant chase will result in the same lower Hessenberg block structure fill in $U$ but each of these blocks will be separated by $u-1$ columns (not $u$). When we reduce the first row subsequently, the plane rotations will be between columns $(u+1) \ldots 2$ of $A$. Note that column $u+1$ of $A$ should have been involved in the chase to reduce the fill generated from reducing the entry $A(1, 2)$ (as $l > 0$). When we reduce the entry in $A(1, u+1)$ and chase the resultant fill, applying the plane rotation to $U$ will result in fill of size $(l+2)$ in the lower triangular part of $U$. This is due to the fact that the fill from the existing Hessenberg blocks in $U$ of order $l+1$ will result in new fill in $U$ as column $u+1$ in $A$, is part of the chase when reducing the first column of $A$ and part of the actual reduction when reducing the entries in first row of $A$. After applying $u-1$ such plane rotations, that fully reduce the first row to bidiagonal, to $U$, we would have a lower Hessenberg block of order $(l+u)$. Thus after reducing the first column and row and chasing the resultant fill in $A$, $U$ will be a block diagonal matrix where the first block will be (l+1)-by-(l+1) and the rest of the blocks (l+u)-by-(l+u) all lower Hessenberg.

Figure 10(a) uses an example $(6, 4)$-band matrix $A$ of the size 16-by-16. Figure 10(b) shows the fill pattern in $U$ after reducing the first column of $A$ and resultant chase. Only the first block of the fill in $U$ is numbered for the corresponding entries that caused it. Figure 10(c) shows the fill pattern in $U$ after reducing the first row of $A$ to bidiagonal and the resultant chase. All the new fill is marked with $\bullet$.

Each subsequent reduction of a column and a row pair will result in a block fill of size (l+u)-by-(l+u-1) in $U$ in the lower triangular part, except the first block which will be of size (l+u)-by-l. The fill in the upper triangular part of $U$ will still be one super diagonal for a every column and row pair. The structure of the fill in $U$ will be similar to figure 8(c) except the size of the blocks will be larger.

**Unsymmetric $A$, $r > 1$:** When $r > 1$ the reduction algorithm leaves $r$ diagonals in the upper and lower triangular part, whereas when $r = 1$ above we will reduce the matrix to bidiagonal in one iteration, which is equivalent to leaving $r-1$ diagonals in the lower triangular part. This change is required to avoid the one column interaction between the plane rotations of the lower and upper triangular part as explained above in $r = 1$ case, so that the fill pattern in $U$ remains simple. Without this small change, the fill pattern in $U$ is still predictable, but will be vastly different from the previous three cases. We will not discuss it in this paper.

Applying the plane rotations from the reducing and chasing the fill of first $r$ columns of $A$, to $U$, will result in block diagonal matrix $U$, where each block is $(l-r, r)$-band matrix of the order $l$. Each of this blocks will be separated by a distance of $u$. The next set of plane rotations, to reduce the entries in $r$ rows in the upper triangular part of $A$, will cause fill in $U$ in the $u$ columns between the existing Hessenberg blocks in $U$. The block fill will be a $(u-r, r)$-band matrix of the order of $u$. Thus after reducing the first $r$ column and rows $U$ will be a block diagonal with two different type of blocks as described above alternating in the diagonal. Figure 11(b)-(c) shows the fill pattern after reducing the entries in the first $r$ columns and rows respectively. The first block of fill is numbered for the corresponding entries that created the fill in both the figures.

Reducing $r$ columns in $A$ after reducing the first $r$ columns and rows will create a block fill in the lower triangular part of $U$ that consist of a rectangular block of the size (u-r+1)-by-(l-r) and below that a block of $(0, l-r)$-band matrix of size (r-1)-by-(l-1) with a zero diagonal. Figure 11(e) shows the fill while reducing the entries from the third and fourth columns. Reducing $r$ rows after the first $r$ rows will create a block fill in the lower triangular part of $U$ that consist of a rectangular block of the size (l-r+1)-by-(u-r) and below that a block of $(0, u-r$-band matrix of size (r-1)-by-(u-1) with a zero diagonal. Figure 11(e)-(f) shows the fill while reducing the entries from the third and fourth columns and rows.

From the above description of the fill for the four cases, it is straight forward to keep track of the first and last row of any column of $U$. For any given column $k$, the first row in $U$ to which a plane rotation that reduced an entry in row $i$ and the plane rotations to chase its fill will be applied is $k-i$. This holds true in all four cases above. For keeping track of the last row for any column in $U$, we only need to keep track of the last row of the first block of $U$ and $V$ and update it after reducing $r$ rows/columns based on the above cases. While reducing a particular set of $r$ column and rows and chasing the fill it is simple to derive the last row for a given column from the last row of the first block based on the above cases. We will leave the indexing calculations out of the paper for a simpler presentation.

## 4.1 Flops

Let us consider the case of reducing $(l, u)$-symmetric band matrix of order $n$ to tridiagonal form again. The number of plane rotations required to reduce the symmetric matrix using Schwarz's diagonal reduction method is

$$g_d = \sum_{d=3}^{b} \sum_{i=d}^{n} 2 + 2 \left\lfloor \frac{\max(i-d, 0)}{d-1} \right\rfloor$$

where $b = u+1$. The number of plane rotations required to reduce the symmetric matrix using Schwarz's row reduction method is

$$g_r = \sum_{i=1}^{n-2} \sum_{d=3}^{\min(b, n-i+1)} 2 + 2 \left\lfloor \frac{\max(i-d, 0)}{b-1} \right\rfloor$$

For finding the number of of floating point operations for our blocked reduction, when block size is $r$-by-$c$ we will use the two step approach as before to get

$$g_{b_1} = \sum_{i=1}^{n-r-1} \sum_{d=r+2}^{\min(b,n-i+1)} 2 + 2 \left\lfloor \frac{\max(i-d,0)}{b-1} \right\rfloor$$

$$g_{b_2} = \sum_{i=1}^{n-2} \sum_{d=3}^{\min(r+1,n-i+1)} 2 + 2 \left\lfloor \frac{\max(i-d,0)}{r} \right\rfloor$$

$$g_b = g_{b_1} + g_{b_2}$$

Asymptotically, number of rotations in all the three methods will be the same. But there will be a significant difference based on the constant factor. Let us assume we will not exploit the structure of $U$. Then every one of the plane rotation must be applied to one column of $U$ each of size $n$. Reducing the number of plane rotations will play a significant role in reducing the flops for accumulation of the plane rotations.

Given a 1000-by-1000 matrix, and $l = 150$. Schwarz's row reduction method will require only 22% of plane rotations as of diagonal reduction method and 57% plane rotations as blocked reduction method for the default block size. When $l = 999$ the row reduction method will require only 17% of the plane rotations as the diagonal reduction method and 52% of the plane rotations as the blocked method. The increase in the number of plane rotations for the blocked method is due to the rotations to reduce the $r$ diagonals as a second iteration($ng_{b2}$ above). These rotations are especially costly as we will be applying them to a full $U$.

We can exactly mimic the Schwarz's row reduction and still do blocking by choosing $r = 1$, leading to a 50% reduction in the flop count. We should note that the cost of small increase in the flops when we reduce $A$ without the accumulation was offset by the blocking. But in this case, when the flops almost doubles, we ensure that we will $r = 1$ in the default block size. Unless there are some compelling reasons, say improvements like blocking the accumulation in $U$ itself which offsets the cost in flops $r > 1$ case will be practically slower when $U$ and $V$ are required.

## 5. PERFORMANCE RESULTS

We compare our band reduction routine piro_band_reduce against that of SBR's driver routine `DGBRDD` [Bischof et al. 2000a] and LAPACK's symmetric and unsymmetric routines :`dsbtrd` and `DGBBRD` [Anderson et al. 1999] in this section. All the tests were done with double precision arithmetic with real matrices. The performance measurements were done on a machine with 16 dual core 2.2 GHz AMD Opteron processors, with 64GB of main memory. PIRO_BAND was compiled with `gcc 4.1.1` with the options `-O3`. LAPACK and SBR are compiled with `g77` with the options `-O3 -funroll-all-loops`.

Figure 12 shows the raw performance of our algorithms with the default block size used by our routines. We estimate the default block size based on the problem size. We do not adjust the block size based on hardware or compiler parameters. Performance of our algorithms depend to some extent on the block size. The optimal block sizes can vary depending on the architecture, compiler optimizations

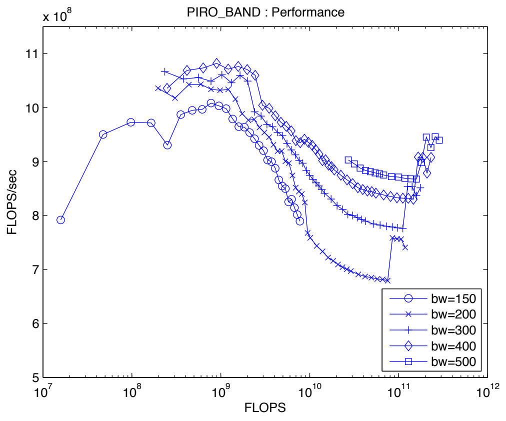

Fig. 12. Performance of the blocked reduction algorithm with default block sizes

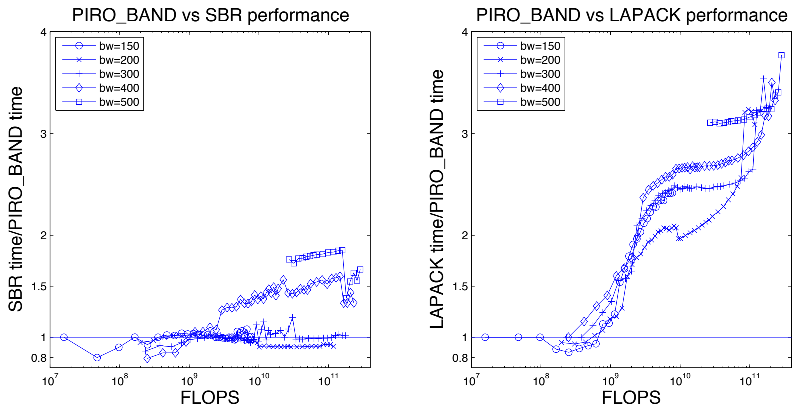

Fig. 13. Performance of piro_band_reduce vs SBR and LAPACK

and problem size. The default block size is not the optimal block size for all the problems, even in this particular machine. Nevertheless, we will use the results from default block size for all the performance comparisons in this section. The effects of the different block sizes on the performance are discussed in the companion paper [Rajamanickam and Davis 2009].

Figure 12 shows the effect of blocking for different bandwidths. The tests were run on symmetric matrices of the order 200 to 10000 with bandwidths 150 to 500. As the bandwidth increases the performance of our algorithm is stable. For smaller bandwidths, we can clearly see the effect of cache: helping the performance for smaller matrices and then negatively affecting the performance as the problem size increases. But blocking stabilizes the performance even for smaller bandwidths as the problem size increases.

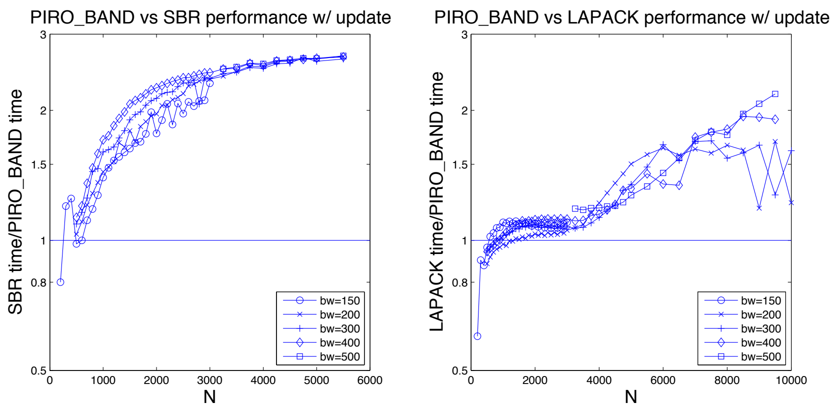

Fig. 14. Performance of piro_band_reduce vs SBR and LAPACK with update

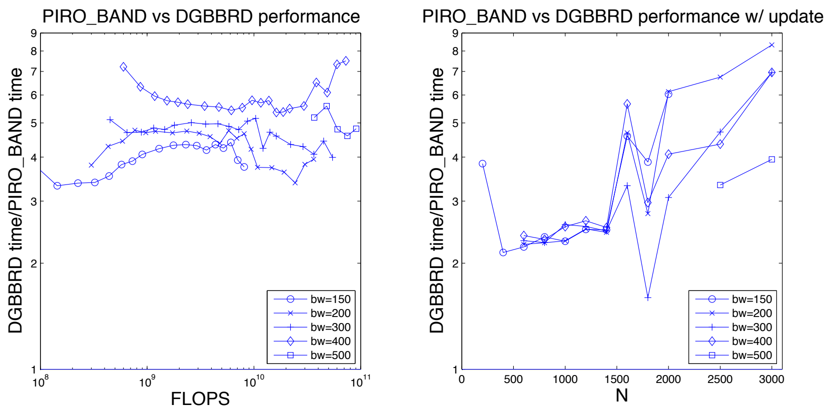

Fig. 15. Performance of piro_band_reduce vs DGBBRD LAPACK

Figure 13 compares the performance of our blocked algorithm and SBR's band reduction routines, without the accumulation of plane rotations. We will provide SBR to the maximum workspace it requires $n \times u$ for a matrix of size $(u, u)$-band matrix of order $n$ and let it choose the best possible algorithm. For smaller matrices, SBR is about 20% faster than our algorithm. When the problems get larger, for smaller bandwidths, the performance of both the algorithms are comparable even though the work space required by our algorithm is considerably smaller.(32-by-32). As the bandwidth increases our blocked algorithm is about 25% to approximately 2x faster than SBR algorithms while using only fraction of the work space.

The results of the performance comparison against LAPACKS's `dsbtrd` is shown in figure 13. LAPACK's routines does not have any tuning parameters except the size $n$ workspace required by the routines.`DSBTRD` performs slightly better than our algorithm for smaller problem sizes. As the problem sizes increase, our algorithm is faster than `dsbtrd` even for smaller bandwidths. The performance improvement will be in the order of 1.5x to 3.5x. `dsbtrd` uses the revised algorithm as described in [Kaufman 1984].

When we accumulate the plane rotations we will use one entire row of the symmetric matrix as our block as this cuts the floating point operations required by nearly half. The performance of our algorithm against SBR and LAPACK algorithms are shown in the figures 14 respectively. Our algorithm is faster than SBR by a factor 2.5 for larger matrices. It will be 20% to 2 times faster than 2 times faster than LAPACK's `dsbtrd` implementation. As in the no accumulation case, the larger the bandwidth the better the performance improvement for our algorithm.

We compare our algorithm against LAPACK's `DGBBRD` for the unsymmetric case. This does not use the revised algorithm of Kaufman yet. So it can be made better. The results when we compare our algorithm againest the existing version of `DGBBRD` are in figures 15 when there without and with the accumulation of plane rotations respectively. Our algorithm is about 4 to 7 times faster in the former case and about 2 to 8 times faster in the later case.

## 6. CONCLUSION

We presented a blocked algorithm for band reduction that will generate no more fill than any scalar band reduction method using pipelining. The work space required for our algorithm is much smaller, equal to the size of the block, than the work space required by competing methods. Our blocked methods perform slightly more work than the Schwarz's row reduction method but less than both LAPACK and SBR. We also presented the algorithm to accumulate the plane rotations while utilizing the non zero structure of $U$ and $V$ for our blocked algorithm. Our algorithm will reduce the number of plane rotations by half, hence the number of flops in accumulations by half, than LAPACK's accumulation algorithm. When the plane rotations are not accumulated, our algorithm is about 2 times faster than SBR and 3.5 times faster than LAPACK. When the plane rotations are accumulated the speed up is about 2x and and 2.5x against LAPACK and SBR. Our blocked algorithm manages to balance all the three requirements, cache utilization, number of floating point operations and work space requirement. The ideas here can be extended to other plane rotation based reduction techniques, for example bidigonalization of a sparse matrix.

## REFERENCES

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. 1999. *LAPACK Users' Guide*, Third ed. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Bischof, C. H., Lang, B., and Sun, X. 2000a. Algorithm 807: The SBR toolbox—software for successive band reduction. *ACM Trans. Math. Softw. 26,* 4, 602–616.

Bischof, C. H., Lang, B., and Sun, X. 2000b. A framework for symmetric band reduction. *ACM Trans. Math. Softw. 26,* 4, 581–601.

Givens, W. 1958. Computation of plane unitary rotations transforming general matrix to triangular form. *Journal of the Society of Industrial and Applied Mathematics. 6,* 1, 26–50.

Golub, G. H. and VanLoan, C. F. 1996. *Matrix Computations*, 3rd ed. The Johns Hopkins University Press.

KAUFMAN, L. 1984. Banded eigenvalue solvers on vector machines. *ACM Trans. Math. Softw. 10,* 1, 73–85.

KAUFMAN, L. 2000. Band reduction algorithms revisited. *ACM Trans. Math. Softw. 26,* 4, 551–567.

LANG, B. 1993. A parallel algorithm for reducing symmetric banded matrices to tridiagonal form. *SIAM J. Sci. Comput. 14,* 6, 1320–1338.

MURATA, K. AND HORIKOSHI, K. 1975. A new method for the tridiagonalization of the symmetric band matrix. *Information processing in Japan 15*, 108–112.

RAJAMANICKAM, S. AND DAVIS, T. A. 2009. PIRO_BAND: Pipelined plane rotations for blocked band reduction. *Submitted to ACM Trans. Math. Softw.*.

RUTISHAUSER, H. 1963. On Jacobi rotation patterns. *Proceedings of Symposia in Applied Mathematics 15*, 219–239.

SCHWARZ, H. R. 1963. Algorithm 183: Reduction of a symmetric bandmatrix to triple diagonal form. *Communications of the ACM 6,* 6, 315–316.

SCHWARZ, H. R. 1968. Tridiagonalization of a symmetric band matrix. *Numer. Math. 12,* 4, 231–241.

VAN LOAN, C. AND BISCHOF, C. 1987. The WY representation for products of householder matrices. *SIAM J. Sci. Stat. Comput. 8,* 1.